PRACTICAL PROGRAMMING IN BASIC
Study Unit 2
24702-2
Ed 2

AN 'INSIDE' VIEW OF COMPUTERS

Printed in South Africa

SCHOOL OF COMPUTER TRAINING

PROGRAMMING IN BASIC
STUDY UNIT 2

AN "INSIDE" VIEW OF COMPUTERS

Edition 2

STUDY UNIT 2

YOUR LEARNING OBJECTIVES

WHEN YOU COMPLETE THIS UNIT, YOU WILL BE ABLE TO:

☐ Identify the major components of the Central or Microprocessing Unit . . . . **Pages 1–2**

☐ Perform simple arithmetic operations with hexadecimal and binary numbers (the numbering systems your computer uses) . . . . **Page 3**

☐ Convert decimal, hexadecimal and binary numbers . **Pages 3–11**

☐ Understand how your computer uses "bytes" to represent letters of the alphabet . . . . **Page 12**

☐ Distinguish between bits and bytes . . . . **Pages 12–14**

☐ Explain what is meant by 16K or 64K RAM . . . . **Page 15**

☐ Describe how compilers and interpreters perform an essential role in data processing . . . . **Pages 15–16**

☐ Understand how simple programs and data are entered and processed by a microcomputer . . . . **Pages 17–19**

STUDY UNIT 2
AN "INSIDE" VIEW

DO YOU KNOW?

- What the "brain" of the computer is?
- The numbering system computers use?
- How and where data is stored in computer memory?

LOOKING INSIDE THE MICRO

In the preceding Unit we explored the past, present, and future of the computer revolution; from the immense to the microscopic; from huge mainframe computers to the microprocessor chip small enough to pass through the eye of a sewing needle. The dimension of time also showed an almost infinite variation. Just think of the many years of human existence it took to develop today's computers; contrast that speed with the billionths of a second at which some of those computers work.

As interesting as the history of the computer revolution is, the present and future hold even greater excitement.

In this lesson, we will look closer at the microcomputer and its various components and attachments. We will see how the processor or brain of the microcomputer works and how it controls other machines.

The brain of the microprocessor is actually composed of a set of integrated circuits (IC's) that control the arithmetic, logic, storage and other devices of the computer system. The most important circuit in a microcomputer is the Central Processing Unit (CPU) or Microprocessing Unit (MPU). The CPU is divided into two parts: the *Arithmetic-Logic Unit* (ALU) contains the wiring to do mathematical functions and comparisons. The *Control Unit* interprets instructions and regulates the movement of data through computer storage.

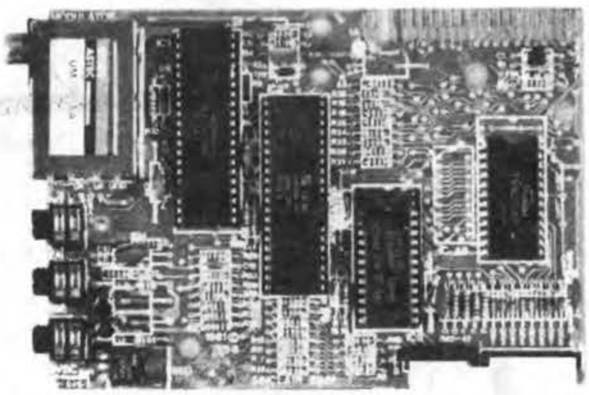


FIGURE 1—The microprocessor of the Timex Sinclair is hidden beneath the keyboard.

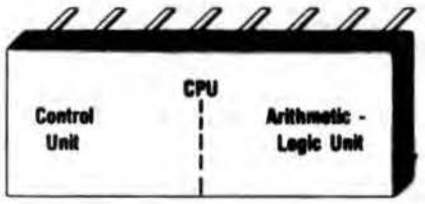


*FIGURE 2—**The Central Processing Unit (CPU) is the same as Microprocessing Unit. It is usually in the form of integrated circuits on a "chip" of silicon mounted on a pinboard. The pinboard can be as small as a dime or as large as a bar of soap. The pinboard module can be removed and replaced within minutes, eliminating the need for costly repairs.***

COMPUTER MEMORY

There are two different types of memory within the computer itself: ROM and RAM.

The ROM (Read Only Memory) integrated circuit contains the operating system which is a program permanently wired into the chip. It directs the CPU to do all of its various functions. It is through its ROM that the computer can understand program instructions written in BASIC. ROM can interpret BASIC statements and translate them into the machine's own internal language.

The RAM (Random Access Memory) is the "scratch pad" of computer memory. In it we can store our programs and data; run them and then enter a whole new set of data and run the program again. Information in RAM is accessible by the CPU at speeds many times faster than one second. One of the main differences between ROM and RAM is that ROM is permanently programmed while RAM memory can be changed at any time the programmer desires.

DATA STORAGE

Both types of computer memory are composed of little more than a series of discrete locations known as BITS. A BIT is a Binary Digit. BITS are capable of having one of two possible values. These two opposite values are sometimes called "HOT" or "COLD"; "ON" or "OFF"; "1" or "Ø."

Computer engineers must be able to "THINK" the same way that the computer does—that is, everything in storage is nothing more than a series of bits, each one being either "ON" or "OFF." In order to do this, a numbering system called BINARY is used.

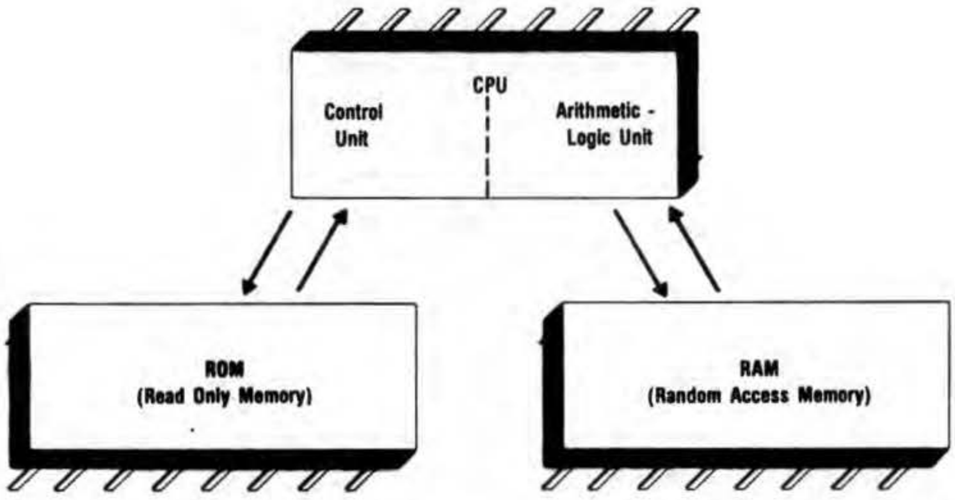


*FIGURE 3—**ROM is built into the computer by the manufacturer. RAM is the "scratch pad" memory which can be erased.***

NUMBERING SYSTEMS

A numbering system is nothing more than a means for expressing numbers. Almost all human cultures use the decimal system, probably because we have ten fingers. This system of counting by tens has become second nature to us. The computer, however, has to do all of its counting by two's. Before we tackle the binary system, let's take a closer look at our own numbering system.

The Decimal System

The decimal system can also be referred to as Base 1Ø. In all numbering systems, two concepts are of primary importance:

1. Every numbering system has a set of allowable numbers.

2. Every numbering system uses *positional* values to represent numbers greater than the largest single digit number.

In Base 1Ø, we can use ten different values in any one position. They are Ø,1,2,3,4,5,6,7,8, and 9. But we have to express values larger than 9. To do this we write large numbers using more than one position.

For example, the number of days in a year is 365. But what does this number really mean? If this is a decimal number, we could say that this number is really three hundreds, six tens, and five units.

Each of the positions in a decimal number is ten times greater than the previous position. Using this numbering scheme there is no limit to the values we can write:

1ØØ
1,876,45Ø,139
1.2ØØ1Ø47
52958.766Ø

To Review

1. In any one position of a decimal number, we can write one of ten different values.
2. We can express numbers larger than nine by defining positions each ten times larger than the preceding position.

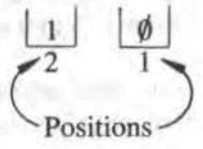
Although you may seldom wish to "think" in computer language, it is worthwhile to understand how this binary numbering system operates. In fact, when you want to create or modify programs, this knowledge will be excellent background. So, let's see how the computer is able to function by using only zeros and ones.

The Binary System

In binary, we only have two possible values with which to work. And yet, in binary, as in any numbering system, there is no limit to the number of things we can describe. How can this be done?

In the decimal system, we can define positions to express numbers larger than the largest one-digit value. But in binary each position is *two* times greater than the previous position (as opposed to ten times greater in decimal).

If we have to write the value that represents "nothing," we would show that as "Ø" in binary, the same as in decimal. The same is true for the value "1." But, while in Base 1Ø we can express a value one unit larger than one as "2," there is no such symbol in binary. We can use only "Ø" and "1." So in binary we must use two positions to write this value. This would be shown as 1Ø (binary). That is, no value in the first unit position and a value of one in the two's position. The programmer writes out "2" as 1Ø or in the form below:



The more positions you add (from right to left), the larger the binary value. Let us look at some of the various position values in binary.

. . . 128 64 32 16 8 4 2 1

Note that as we move from right to left, each position is two times greater than the previous position.

This progression of doubling the value of each position has no limit—giving us the ability to express very large numbers.

. . .	8192	4Ø96	2Ø48	1Ø24	512	256	128	64

Converting Binary to Decimal. Suppose we were given a binary number "1ØØØ111Ø"—what is its value in decimal? First, determine the position value of each digit. Write the values underneath each digit. The binary numbers now look like this:

1	Ø	Ø	Ø	1	1	1	Ø
128	64	32	16	8	4	2	1

Now, add the position values where a "1" appears. That is, 128 + 8 + 4 + 2 = 142 (decimal). Let's try another: Ø11Ø11Ø1. The position values would be displayed as follows:

Ø	1	1	Ø	1	1	Ø	1
128	64	32	16	8	4	2	1

Adding all the "1" positions together, you have

64 + 32 + 8 + 4 + 1 = 1Ø9 (decimal)

Some interesting characteristics of binary numbers may now be mentioned.

- It usually takes more digits to write a number in binary than it does in decimal.
- Any binary number ending in a "1" will be converted into an odd number in decimal.

Now, pause for a moment and complete the Programmer's Check on the next page. Write your answers or do your calculations in the space provided. Then, check your answers with those on the page specified.

Now, suppose we wanted to convert decimal numbers to binary? It is just a matter of practice.

Decimal to Binary Conversion. There are several ways of converting from decimal numbers into their binary equivalents. One method is to write out the position values until the leftmost position is equal to or just less than the decimal number you wish to convert. Working from left to right, put a "1" into the first position and subtract the position value from the decimal number. Now, do the same for the remainder until you wind up with nothing left.

For example, let's try converting the decimal number "84" into binary. First, write out the positions.

64	32	16	8	4	2	1

Put a "1" in the 64's position and subtract 64 from 84.

1						
64	32	16	8	4	2	1

We have 2Ø left. Put a "Ø" in the 32's position and subtract nothing, since 2Ø is less than 32.

1	Ø					
64	32	16	8	4	2	1

There is a 16 in 2Ø, so we now put a "1" in the 16's position and subtract, leaving 4.

1	Ø	1				
64	32	16	8	4	2	1

Put a "Ø" in the 8's position because we only have 4 left.

1	Ø	1	Ø			
64	32	16	8	4	2	1

By now entering a "1" in the 4's position and subtracting, we have Ø left. Fill the remaining positions with Ø's and you're done.

1	Ø	1	Ø	1	Ø	Ø
64	32	16	8	4	2	1

PROGRAMMER'S CHECK

1

Converting Binary Numbers to Decimals

Convert the following numbers from binary to decimal. Use the note pad below to perform the calculations. As you compute, you are performing the operations similar to what RAM does within a computer. Check your answers before proceeding.

										Answer
1.	ø	1	ø	ø	ø	1	ø	1	=	
2.	1	ø	ø	ø	ø	1	ø	1	=	
3.	1	1	ø	1	1	ø	1	1	=	
4.	ø	1	1	ø	ø	ø	ø	1	=	
5.	1	ø	ø	ø	ø	1	ø	ø	=	
	128	64	32	16	8	4	2	1		

(Answers on Page 6)

NOTE PAD

PROGRAMMER'S CHECK ANSWERS

1

1.	69	4.	97
2.	133	5.	132
3.	219		

We are left with the binary number 1Ø1Ø1ØØ. To check our answer, we can convert into decimal by adding together all the "1" position numbers, 64 + 16 + 4 = 84 (decimal), and see we were correct.

Let's do another. What is the binary number for 56? Since 56 is greater than position 32, place a "1" there and subtract the remainder.

	1					
64	32	16	8	4	2	1

Since 24 is greater than 16, place another "1" there and subtract.

	1	1				
64	32	16	8	4	2	1

The remainder "8" is equal to position 8, so another "1" is placed there.

	1	1	1			
64	32	16	8	4	2	1

What is placed in the remaining three positions? Yes, "Ø" is placed at positions 4,2, and 1. Now pause a moment and see how you can apply what you've learned. Complete the Programmer's Check on the next page.

Let's look at another method for decimal to binary conversions. In this method we take our decimal number and repeatedly divide it by two, storing the remainder as either "Ø" or "1"—continuing until we are left with Ø. The remainder, in reverse order, will be the binary equivalent. Let's try to convert decimal 84 into binary using this method.

Divide 84 by two: $$\begin{array}{r} 42 \\ 2\overline{)84} \\ \underline{84} \\ \emptyset \end{array}$$

The remainder is: Ø

Divide 42 by two: $$\begin{array}{r} 21 \\ 2\overline{)42} \\ \underline{42} \\ \emptyset \end{array}$$

Again, the remainder is: Ø

Divide 21 by two: $$\begin{array}{r} 1\emptyset \\ 2\overline{)21} \\ \underline{2\emptyset} \\ 1 \end{array}$$

This time, the remainder is: 1

Divide 1Ø by two: $$\begin{array}{r} 5 \\ 2\overline{)1\emptyset} \\ \underline{1\emptyset} \\ \emptyset \end{array}$$

Remainder: Ø

Divide 5 by two: $$\begin{array}{r} 2 \\ 2\overline{)5} \\ \underline{4} \\ 1 \end{array}$$

Remainder: 1

Divide 2 by two: $$\begin{array}{r} 1 \\ 2\overline{)2} \\ \underline{2} \\ \emptyset \end{array}$$

Remainder: Ø

Divide 1 by two: $$\begin{array}{r} \emptyset \\ 2\overline{)1} \\ \underline{\emptyset} \\ 1 \end{array}$$

Remainder: 1

Now put the remainder together, but from bottom to top: 1Ø1Ø1ØØ

PROGRAMMER'S CHECK

2

Converting Decimals to Binary

Convert the following numbers from decimal to binary. Use the scratch pad below to perform the calculations. Check your answers before proceeding.

Answer

1.	**52**	=								
2.	**175**	=								
3.	**83**	=								
4.	**29**	=								
5.	**130**	=								
			128	**64**	**32**	**16**	**8**	**4**	**2**	**1**

(Answers on Page 9)

NOTE PAD

Can you get the same results using this method as you did the previous way? Certainly. Do the Programmer's Check and prove you can obtain the same answers as on the previous Programmer's Check.

PROGRAMMER'S CHECK

3

Converting Decimals to Binary by Dividing by Two

Convert the following numbers from decimal to binary using the division method. Use the scratch pad below to perform the calculations. Check your answers before proceeding.

Answer

1.	**52**	=	☐	☐	☐	☐	☐	☐	☐	☐
2.	**175**	=	☐	☐	☐	☐	☐	☐	☐	☐
3.	**83**	=	☐	☐	☐	☐	☐	☐	☐	☐
4.	**29**	=	☐	☐	☐	☐	☐	☐	☐	☐
5.	**130**	=	☐	☐	☐	☐	☐	☐	☐	☐
			128	**64**	**32**	**16**	**8**	**4**	**2**	**1**

(Answers on Page 10)

NOTE PAD

$2\overline{)52}$

PROGRAMMER'S CHECK ANSWERS

2

52	=			1	1	Ø	1	Ø	Ø
175	=	1	Ø	1	Ø	1	1	1	1
83	=		1	Ø	1	Ø	Ø	1	1
29	=				1	1	1	Ø	1
13Ø	=	1	Ø	Ø	Ø	Ø	Ø	1	Ø
		128	64	32	16	8	4	2	1

The following chart shows binary numbers Ø through 16 and their decimal "partners."

BINARY/DECIMAL PARTNER CHART

BINARY	DECIMAL
Ø	Ø
1	1
1Ø	2
11	3
1ØØ	4
1Ø1	5
11Ø	6
111	7
1ØØØ	8
1ØØ1	9
1Ø1Ø	1Ø
1Ø11	11
11ØØ	12
11Ø1	13
111Ø	14
1111	15
1ØØØØ	16

FIGURE 4—Notice how binary develops a pattern of ones and zeros as the decimal numbers increase. Every combination is used before the digits are expanded. From the above pattern, can you identify the binary partner of 17? If in doubt about your answer, use your conversion skills.

The two ways of converting decimals to binary give you the option of using either subtraction or division to obtain the result. If you wanted to do it, you could construct your own conversion table and maintain the conversion partners for hundreds of numbers. But...

Stop and compare decimal and binary numbers for a moment. The decimal 16 is 1ØØØØ in binary. It takes only two digits to write the value "16" in decimal while it takes five positions in binary. The larger the decimal value gets, the more dramatic this discrepancy gets. Dealing with large binary numbers can get quite cumbersome. Imagine converting the decimal 5,ØØØ,35Ø into binary! Sure, it could be done, but the number would result in a colossal line of zeros and ones which would be rather unmanageable. Here is the result of 5,ØØØ,35Ø being converted:

1	Ø	Ø	1	1	Ø	Ø	Ø	1	Ø	Ø	1	1	Ø	Ø	1	Ø	Ø	1	1	1	1	Ø
41943Ø4	2Ø97152	1Ø48576	524288	262144	131Ø72	65536	32768	16384	8192	4Ø96	2Ø48	1Ø24	512	256	128	64	32	16	8	4	2	1

While converting small decimals into binary worked out fine for computer programmers, larger numbers created giant headaches. After all, a programmer must be able to deal with any numbers—especially when creating programs for space exploration and computing distances among stars. So, a "shorthand" numbering system is also used by programmers. One popular shorthand is called hexadecimal.

The Hexadecimal Numbering System

The word "hexadecimal" stands for "16." This numbering system (commonly referred to as "hex") is actually Base 16. Remember, in Base 2 (binary), we have positions two times larger than the preceding position (from right to left) and two possible numbers—"Ø" and "1." In Base 1Ø (decimal) each position is ten times larger and uses ten numbers "Ø" through "9."

PROGRAMMER'S CHECK ANSWERS

3

52 =			1	1	ø	1	ø	ø
175 =	1	ø	1	ø	1	1	1	1
83 =		1	ø	1	ø	ø	1	1
29 =				1	1	1	ø	1
13ø =	1	ø	ø	ø	ø	ø	1	ø
	128	64	32	16	8	4	2	1

As perhaps you have guessed, in Base 16 (hex), each position is 16 times larger and we use 16 different numbers.

The position in hex looks like this:

16x16x16 = 4ø96	16x16 = 256	16x1 = 16	1

(NOTE: In any numbering system, the right-most position has a position value of "1.")

Now that we have determined the position values, we have to list the numbers that hex uses. The first ten are easy: "ø" through "9." But the next six are not so simple. If we used "1ø" through "15," we see that these numbers are written in two positions and we need numbers that can be written in one position. Any single character symbols could have been used, but the first six letters of the alphabet (A through F) were chosen. These letters must be thought of as *numbers* in hex so that: "A" stands for the number "1ø" in one position; "B" for "11"; "C" for "12"; "D" for "13," "E" for "14," and "F" for "15." While this may seem confusing at first, with a little practice you'll be used to it.

Hex to Decimal Conversion. Suppose we are given the following hexadecimal number —"3C." What decimal number is its equivalent?

Just as we did in the binary to decimal method, we first have to determine the position values.

3	C
16	1

Now multiply the number by the position value and add them together. That is: (3 x 16) + (12 x 1) = 48 + 12 = 6ø (decimal).

(Remember that "C" hex is the value "12.")

Try this hex number "A3." You will get "163" decimal (1ø x 16) + (3 x 1) or 16ø + 3 = 163.

Now notice that whereas it takes three positions in decimal to write the value "16ø," it only takes two in hexadecimal ("A3"). In binary, it would take eight positions.

HEX	DECIMAL	BINARY
ø	ø	ø
1	1	1
2	2	1ø
3	3	11
4	4	1øø
5	5	1ø1
6	6	11ø
7	7	111
8	8	1øøø
9	9	1øø1
A	1ø	1ø1ø
B	11	1ø11
C	12	11øø
D	13	11ø1
E	14	111ø
F	15	1111
1ø	16	1øøøø
11	17	1øøø1
12	18	1øø1ø
FF	255	11111111

FIGURE 5—This conversion chart compares hex numbers side by side with their decimal and binary values.

Examine the conversion chart carefully and notice how, with one hexadecimal number it takes four binary numbers. That is why hexadecimal is binary shorthand. A grouping of four binary numbers can always be expressed as a single hexadecimal number. Attempting to convert a large binary number to hexadecimal would be tedious if we first had to convert to decimal. But we don't! Take the following binary number: "1ØØ1ØØ1Ø." Grouping the number by four and using the preceding chart, the hexadecimal equivalent is "92."

$$\underbrace{1ØØ1}_{9} \qquad \underbrace{ØØ1Ø}_{2}$$

Try converting some other binary numbers into hex in the following Programmer's Check. It is a lot easier than trying to find the decimal value first, isn't it?

PROGRAMMER'S CHECK

4

Converting Binary to Hexadecimal

Use the conversion chart to convert the following binary numbers to hex.

Answer

1. **1ØØ1ØØØ1** = ⎣ ⎦⎣ ⎦ ___ ___ = ____
2. **1111Ø11Ø** = ⎣ ⎦⎣ ⎦ ___ ___ = ____
3. **11ØØ1111** = ⎣ ⎦⎣ ⎦ ___ ___ = ____
4. **1Ø11ØØ11** = ⎣ ⎦⎣ ⎦ ___ ___ = ____
5. **1Ø1Ø1Ø1Ø** = ⎣ ⎦⎣ ⎦ ___ ___ = ____

(Answers on Page 12)

There is an almost infinite number of shorthand systems used by computer manufacturers and machine language programmers. OCTAL, for example, is a common shorthand using a Base 8 system. If you are fascinated by these systems and wish to learn more about this special area of computer technology, you can subscribe to one or more of the numerous magazines and newsletters which inform readers about latest number shorthand systems. Meanwhile, you should know that the hexadecimal system is the "shorthand" used in your computer course.

While it may seem that we have just spent a lot of time coping with mathematics, don't worry. You really don't have to become involved with machine languages. However, it is important that you understand how these languages operate in reducing the amount of space required to store numerical data.

As you will recall, our discussion of numbering systems grew out of an examination of data storage. You may think that we have gotten off the path—but this is far from the truth.

Remember that computer memory is largely composed of bits (binary digits). We said if a bit was "ON," we could represent that as a "1." If it's "OFF," as "Ø."

In memory, we need to store all of the numbers, letters of the alphabet, special characters (commas, periods, dollar signs, etc.) and computer instructions. If we group eight consecutive bits of memory into a single unit, we would be able to have 256 different bit patterns—all the way from "ØØØØØØØØ" to "11111111."

If one bit has two possible states, then eight bits in combination have 2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 or (2^8) or 256 variations.

If we give each of the characters and instructions we need to store a unique "bit pattern," then we can represent all the numbers (1Ø), letters (26) and special characters (18 or more) and still have enough left over for instructions and other purposes.

PROGRAMMER'S CHECK ANSWERS

4

1.	1ØØ1	ØØØ1	
	9	1	= 91
2.	1111	Ø11Ø	
	F	6	= F6
3.	11ØØ	1111	
	C	F	= CF
4.	1Ø11	ØØ11	
	B	3	= B3
5.	1Ø1Ø	1Ø1Ø	
	A	A	= AA

BYTES

Most computers group their bits into eights and call each group a byte. Each byte can contain one letter, one number, one special character, or one machine instruction (or part of an instruction, as we shall see later on).

There are many different coding schemes used to represent a character with a unique bit pattern. We will limit our discussion here to only one of them: EBCDIC. Your computer may use another. But the concepts are all the same, and you will have little trouble adjusting.

EBCDIC

EBCDIC (Extended Binary Coded Decimal Interchange Code) is a code used on many different computer systems. Each of the 256 possible variations of eight bit patterns is assigned a unique, coded meaning.

For example, instead of storing the letter "A" as printed in Figure 6, the CPU recognizes the bit pattern 11ØØØØØ1 as representing the letter "A."

If we wished to store the word "COMPUTER" in RAM, it would need to occupy eight bytes—one byte for each letter.

C	O	M	P	U	T	E	R
1	2	3	4	5	6	7	8

Similarly, the number "1824" would take up four bytes—one byte for each digit.

1	8	2	4

Each byte is divided into two portions. The zone portion being the first four bits; the digit portion occupying the last four bits.

ØØØØ	ØØØØ
ZONE	DIGIT

In the assigning of codes to characters, there are some patterns. For example, notice in the EBCDIC chart that the letters of the alphabet "A" through "I" contain a hexadecimal "C" in the zone portion (11ØØ binary) and a number "1" through "9" (ØØØ1-1ØØ1) in the digit portion.

Similarly, the letters "J" through "R" use the hexadecimal "D" in the zone portion and hex "1" through "9" in the digit portion.

That leaves the letters "S" through "Z" to be coded. I'm sure you won't be surprised to find an "E" in the zone portion. But then, there is a slight variation. The digit portion of the bytes of these letters have a hex "2" through "9." (Why is the letter "S" not an "E1" but an "E2?" The reason is slightly obscured by the past; however, it is probably due to the way the original code was used to punch holes into cards.)

THE EBCDIC CODE		
HEX	BINARY	CHARACTER
4Ø	Ø1ØØ ØØØØ	(space)
5B	Ø1Ø1 1Ø11	$
5C	Ø1Ø1 11ØØ	*
6F	Ø11Ø 1111	?
7A	Ø111 1Ø1Ø	:
7B	Ø111 1Ø11	+ +
7C	Ø111 11ØØ	@
7D	Ø111 11Ø1	,
C1	11ØØ ØØØ1	A
C2	11ØØ ØØ1Ø	B
C3	11ØØ ØØ11	C
C4	11ØØ Ø1ØØ	D
C5	11ØØ Ø1Ø1	E
C6	11ØØ Ø11Ø	F
C7	11ØØ Ø111	G
C8	11ØØ 1ØØØ	H
C9	11ØØ 1ØØ1	I
D1	11Ø1 ØØØ1	J
D2	11Ø1 ØØ1Ø	K
D3	11Ø1 ØØ11	L
D4	11Ø1 Ø1ØØ	M
D5	11Ø1 Ø1Ø1	N
D6	11Ø1 Ø11Ø	O (the letter)
D7	11Ø1 Ø111	P
D8	11Ø1 1ØØØ	Q
D9	11Ø1 1ØØ1	R
E2	111Ø ØØ1Ø	S
E3	111Ø ØØ11	T
E4	111Ø Ø1ØØ	U
E5	111Ø Ø1Ø1	V
E6	111Ø Ø11Ø	W
E7	111Ø Ø111	X
E8	111Ø 1ØØØ	Y
E9	111Ø 1ØØ1	Z
FØ	1111 ØØØØ	Ø (the number)
F1	1111 ØØØ1	1
F2	1111 ØØ1Ø	2
F3	1111 ØØ11	3
F4	1111 Ø1ØØ	4
F5	1111 Ø1Ø1	5
F6	1111 Ø11Ø	6
F7	1111 Ø111	7
F8	1111 1ØØØ	8
F9	1111 1ØØ1	9

FIGURE 6—The Extended Binary Coded Decimal Interchange Code ensures that each hex or binary number has only one character meaning.

The decimal numbers are coded with a hex "F" in their zone portions and their actual number in the digit portion. Thus, the number "6" would appear in a byte as an "F6"; the number "32" would occupy two bytes and would be written as "F3F2."

Now, let's see if we can use this chart (it need *not* be memorized, as long as you can refer to the chart when you need it!) to "decode" some data.

Look at the following coded bytes:

$\boxed{C3}$ $\boxed{C1}$ $\boxed{E3}$

From the chart, we can see that "C3" stands for the letter "C"; "C1" for the letter "A"; and "E3" for the letter "T." Now we can see how the word CAT would be coded in RAM if, perhaps, we wanted to store various animal names in our computer.

```
     C3              C1              E3
|1 1 Ø Ø|Ø Ø 1 1| |1 1 Ø Ø|Ø Ø Ø 1| |1 1 1 Ø|Ø Ø 1 1|
      C               A               T
```

Now, pause for a moment and complete the Programmer's Check on the following page. Write your answers or do your calculations in the space provided. Then, check your answers on the page specified.

ADDRESSABILITY

There is one more important characteristic of bytes in computer memory other than its ability to store characters, and that is its addressability.

Each byte of main storage has a unique address or locatable position. The first byte of main storage is located at address "Ø." The last byte of main storage depends on the amount of storage the chip can hold—that is, its storage capacity.

PROGRAMMER'S CHECK

5

Using the EBCDIC Code

Convert the following data into the appropriate code, using the EBCDIC Code Chart.

1. Give the character code for the hex |5B| |F5| |FØ| |F9|

   ____

2. Give the hex code for *SALES* ______________________

3. Give the character code for |111Ø ¦ ØØ1Ø|, |11ØØ ¦ ØØØ1|,
   |11Ø1 ¦ ØØ11|, |11ØØ ¦ Ø1Ø1|. ______________________

4. Give the character code for |D7| |D9| |D6| |C7| |D9| |C1| |D4|

   ____

(Answers on Page 16)

NOTE PAD

SIZE OF STORAGE

The capacity of storage is often known as so many "K" bytes. (For example 2K, 16K, 64K, etc.) K is a common abbreviation for kilo, meaning 1,000 as in kilometers, kilograms, etc. Actually, there are 1,024 bytes to a kilobyte— because of its basis in binary—2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 = 1,ø24 (2^{10}). Thus, 64K actually equals 65,536 bytes.

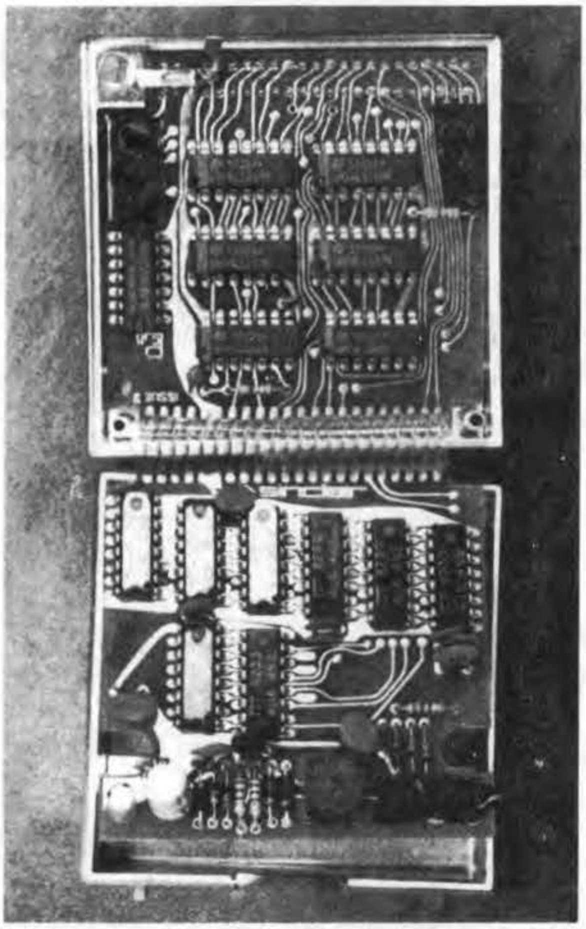


FIGURE 7—This is how the Timex Sinclair 1016 16K RAM module looks on the inside. Each of the more than 16,øøø bytes has its own address.

As you no doubt have noticed, computers possess varying RAM capacities—ranging from 2K up to several hundred. Pocket or briefcase-sized computers may be quite adequate with 8K RAM, but a desk-sized business micro needs a minimum of 64K RAM. Why?

Since each byte of data of program instruction must be stored in a separate internal storage location (address), it takes a great deal of RAM space just to load a business program. An accounting program could take as much as half the available space within RAM, leaving little room for data input and processing results.

Is it essential that you have enough RAM for all the data which is to be processed? A 64K or more RAM memory accommodates most programs and reasonable amounts of data for processing. However, some very large computing tasks require more RAM storage than 64K. Most computers have ports where additional RAM storage can be added. Additional 8K, 16K, 32K, or 64K RAM chips or boards can be added by merely plugging them into the socket.

Once the user realizes that RAM memory is nearly full, the processed data is usually SAVED onto external storage via magnetic tape or on disks. By SAVING onto tape on disks, you can store infinite amounts of data. A single C-90 (90 minute cassette tape) can easily hold 160,000 bytes of data.

COMPILERS

Compilers, or language translators, are software (programs) which are written to translate "higher-level languages" (such as BASIC) into actual machine instructions. Some very small computers actually have a compiler built in.

Machine instructions are retrieved by the control unit one by one from main storage. Each machine instruction gives the CPU the code as to what it is supposed to do (add, subtract, compare or print, etc.) and the storage location (address) of the data it is supposed to manipulate.

Each higher-level language must provide three pieces of information:

1. The type of work to be done (also known as the "operation");
2. the addresses of the data to be "done" (also known as the "operands"); and,
3. *when* this instruction is to be done (also known as its "sequence").

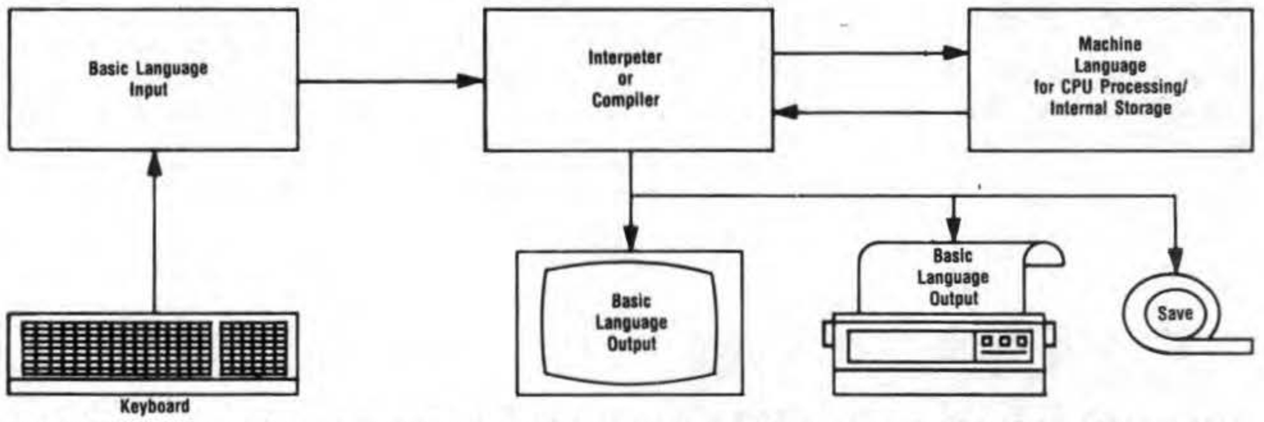


FIGURE 8—A compiler or interpreter is a software program either permanently built into the computer or available as an additional option. This software enables the computer to convert a language such as BASIC into the computer's own electronic or "machine" language. A different type of interpreter is required for different higher languages such as FORTRAN, COBOL or BASIC.

For example, the BASIC instructions to add two values to obtain their sum might look like this:

```
1Ø Let Answer = Number 1 + Number 2
```

This instruction, under control of the BASIC compiler, would be converted into two (or more) machine-level instructions which could look like this:

```
1. ADD 1ØØØ,15ØØ
   Store 2ØØØ
```

The compiler has given each variable within the program (number 1, number 2, answer) an address and has come up with a couple of instructions it can do.

PROGRAMMER'S CHECK ANSWERS

5

1. $5Ø9

2. | E2 | C1 | D3 | C5 | E2 |

3. SALE

4. PROGRAM

When we command the computer to "RUN" this instruction, the following sequence of events occurs.

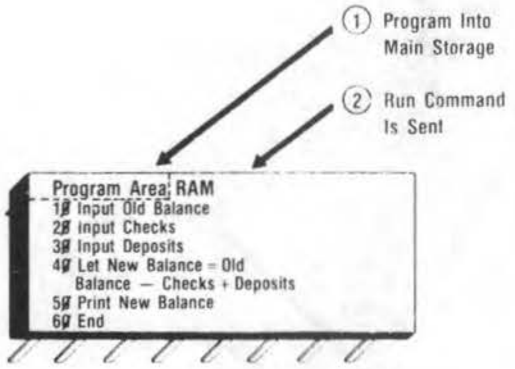
1. The control unit will fetch the first instruction and put it into the instruction register portion of the CPU. The instruction register is no more than a series of bytes which contains one instruction at a time.
2. The ALU (Arithmetic-Logic Unit) is then told to set up its "addition" circuitry.
3. The data at byte 1ØØØ and byte 15ØØ is copied into the ALU.
4. The two values are added together.
5. The next instruction is fetched.
6. The CPU is directed to place the result of the previous calculations into RAM at byte 2ØØØ.

Now, let's take a look at the way an entire program works. This is a small program that might aid us in balancing our checkbook. At the end of each month we want the computer to store our old balance (from last month), subtract the total of our checks, add the sum of our deposits, and store the answer as the new balance.

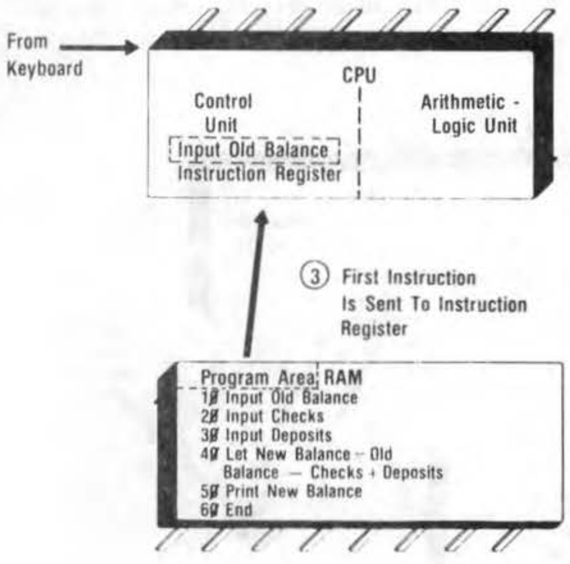
So, we have to create a program that will perform these functions each month automatically. All we'll need to do is to feed in the variable data (checks, deposits, etc.) and let the CPU and program take over. What are the elements of the program? Using BASIC and the computer keyboard, we would input the following:

```
1Ø   INPUT OLD BALANCE

2Ø   INPUT CHECKS

3Ø   INPUT DEPOSITS

4Ø   LET NEW BALANCE = OLD
     BALANCE - CHECKS
     + DEPOSITS

5Ø   PRINT NEW BALANCE

6Ø   END
```

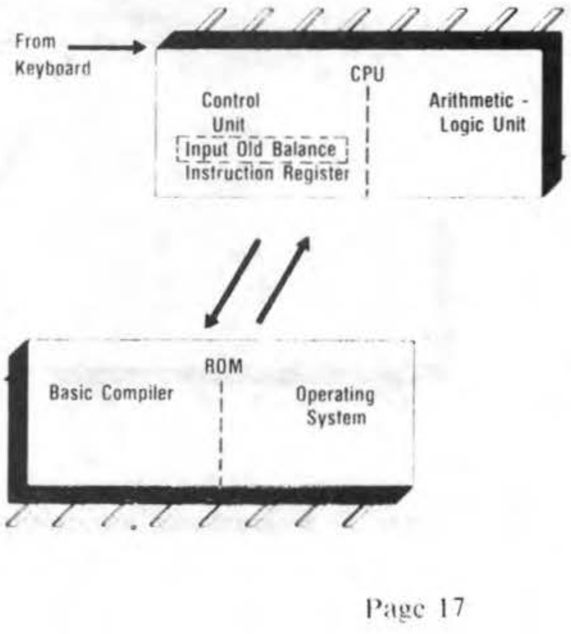
1. Our first task, now, is to enter the program into RAM. Once this program is entered into RAM we can test it to see whether the program instructions are understood by the CPU and will give accurate and reliable results. To do this, the computer is given the command to "RUN" the program.



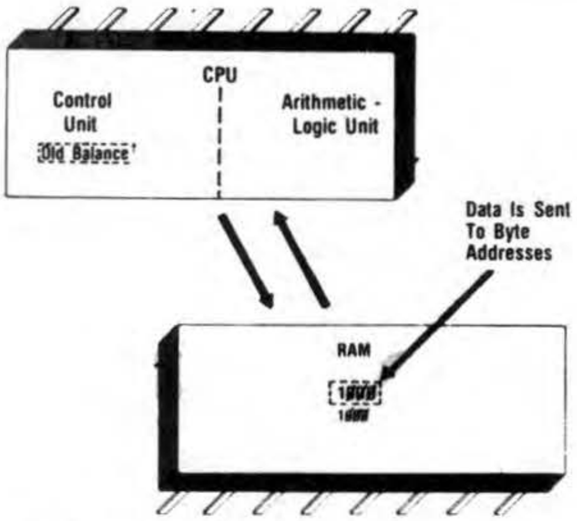
2. The RUN command triggers the CPU Control Unit to retrieve the first instruction and place it in the instruction register.



3. The CPU then sends the instructions to the BASIC interpreter. (The Basic Compiler recognizes this instruction as a program "request" for data that will be stored at an address. So, it sets up a series of bytes which will contain a series of numbers we call the OLD BALANCE. For this example, we will say that it establishes byte number 1ØØØ as the first byte to hold this number.)



4. The CPU instantly sends the "reserved addresses" to RAM.



5. We'll now say that $1ØØØ is sent as the old balance to be used in this program. (In our next lesson, we'll see how this is done.) This value will be stored beginning at byte 1ØØØ.

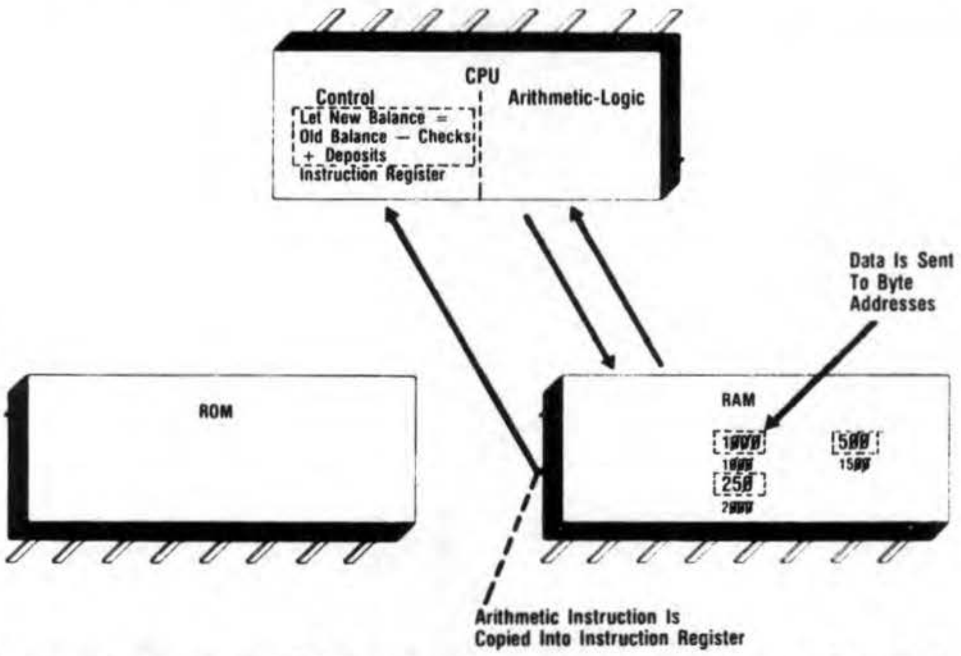
6. The control unit then fetches the next instruction, into the instruction register, and the same method is used to accept a value of $5ØØ as DEPOSITS and stores it at an initial byte of 15ØØ.

7. After the third instruction is fetched and interpreted, we'll say that $25Ø is given as CHECKS and is located starting at byte 2ØØØ.

8. The next instruction, LET NEW BALANCE = OLD BALANCE − CHECKS + DEPOSITS, is now brought into the CPU which now looks like the figure below.

This next instruction is actually translated into a series of machine instructions, which first will copy the values stored at byte 1ØØØ (old balance of $1ØØØ) and byte 15ØØ (deposit of $5ØØ) and place them into the Arithmetic-Logic Unit of the CPU. These areas where numbers are placed for calculations to be done are called "data registers."

9. The circuitry to do addition is already "wired-in" to the ALU and it adds the two values together and holds on to the sum.



CPU
Control
Let New Balance = Old Balance − Checks + Deposits
Instruction Register
Arithmetic-Logic
Data Is Sent To Byte Addresses
ROM
RAM
1ØØØ
1ØØØ
5ØØ
15ØØ
25Ø
2ØØØ
Arithmetic Instruction Is Copied Into Instruction Register

FIGURE 9—Once the program and instructions have been entered, data can be stored or processed.

10. Next, the value at byte 2ØØØ (checks of $25Ø) is placed into a data register and is subtracted from the previous sum.

11. The final step of this instruction is to place the result (NEW BALANCE) back into RAM where it might be addressed at byte number 25ØØ.

12. When the next instruction is put into the instruction unit (the PRINT NEW BALANCE), the value at address 25ØØ (NEW BALANCE) is sent to an output device (such as a printer or a TV screen).

13. The last instruction (END) informs the CPU that the program is ended. At this point one can choose to rerun the program with new values (for example, at the end of next month's checking account period) or to load and run an entirely different program.

This concludes Instructional Unit 2. In this lesson we have taken a closer look at the "brains" of the computer and have begun to appreciate the way in which the computer processes data.

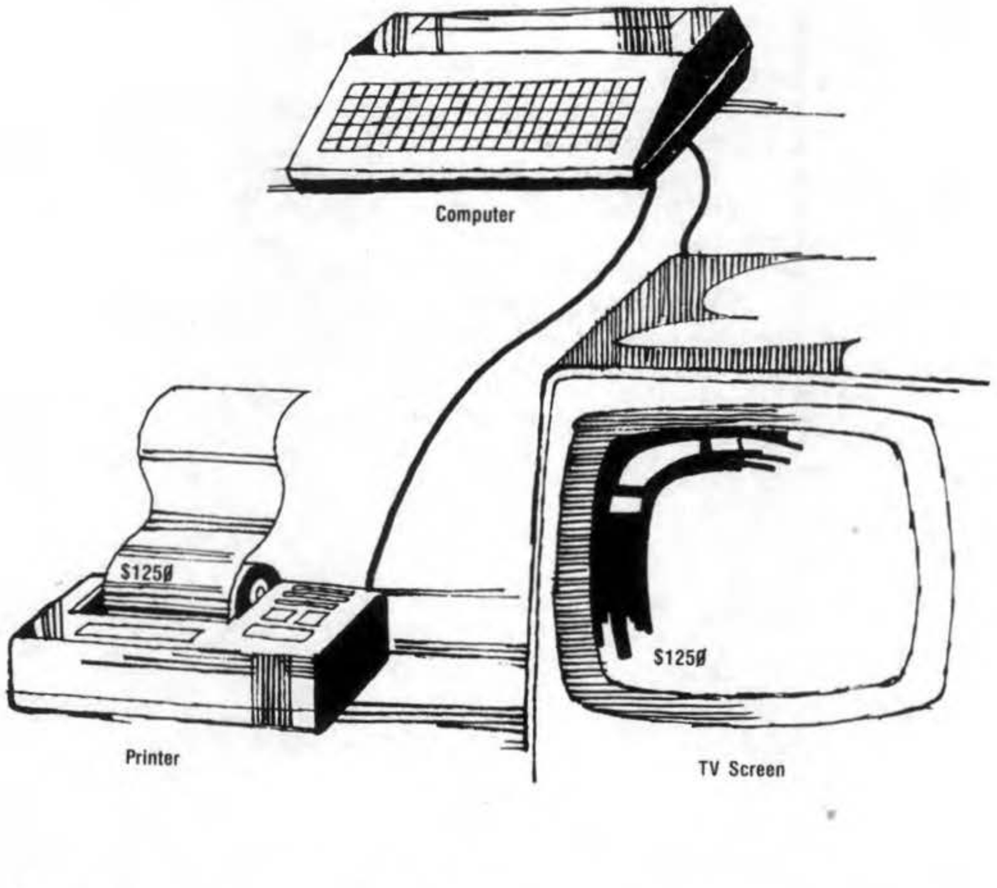


FIGURE 10—When processing is complete, the result is sent to a CRT or to a printer.

DO YOU KNOW NOW?

These were the questions posed at the beginning of the lesson.

- **What the "brain" of the computer is?**
  The brain of the computer is the CPU. The CPU contains the circuitry necessary to process computer instructions and to do arithmetic and comparisons.

- **The numbering system computers use?**
  Computers use the binary numbering system.

- **How and where data is stored in computer memory?**
  Data is stored in both RAM and ROM in groupings of eight bits known as one byte. Data in ROM bytes is permanent; in RAM bytes they are not. Various coding schemes are used to represent data in memory. One of these is EBCDIC.

SCHOOL OF COMPUTER TRAINING

EXAM 2

AN "INSIDE VIEW"

24702-2

Questions 1-20: *Circle the letter beside the one best answer to each question*

1. There are two basic types of computer memory commonly known as

   (a) CPU and ALU.
   (b) ROM and data.
   (c) ROM and RAM.
   (d) integrated circuits and mainframes.

2. Random Access Memory (RAM) is

   (a) a permanently wired integrated circuit.
   (b) the "scratch pad" of computer memory.
   (c) the "brains" of the computer.
   (d) not a part of the computer system.

3. Bits are

   (a) capable of having one of two possible values.
   (b) binary digits.
   (c) the basis of all computer memory.
   (d) all of the above.

4. The decimal value "5" can be expressed in binary as:

(a) 1Ø1Ø	(c) 11ØØ
(b) Ø111	(d) Ø1Ø1

5. The value "11ØØ" in binary is what in decimal?

(a) 8	(c) 12
(b) 1Ø	(d) 14

6. The largest decimal number that can be expressed in binary is:

(a) 128	(c) 64
(b) 1	(d) unlimited

7. Any group of four binary digits can be expressed in one digit in which numbering system?

(a) OCTAL	(c) Decimal
(b) Hexadecimal	(d) Binary

8. How many bits are there in a typical byte?

(a) 4	(c) 8
(b) 6	(d) 1Ø

9. In one byte of main storage we can store

(a) one letter of the alphabet.
(b) one digit of a number.
(c) one special character (comma, period, dollar sign, etc.).
(d) any of the above.

10. The characters represented by the EBCDIC code of F8F3FØ are

(a) the letter E.	(c) the number 83Ø.
(b) the word ICE.	(d) none of the answers given.

11. Now, translate this EBCDIC code: C3D6D4D7E4E3C5 (use the chart). The word it represents is

(a) COMPUTE
(b) CALCULATE
(c) COMPILE
(d) none of the above.

12. What numbering system is a four-bit binary shorthand method?

(a) OCTAL
(b) Hexadecimal
(c) Binary
(d) Decimal

13. The programs which are written to translate "higher-level languages" (such as BASIC) into actual machine instructions are called:

(a) RAM
(b) Compilers
(c) CPU
(d) EBBDIC

14. The "On/Off" or "Hot/Cold" signals understood by a computer are:

(a) 9 and 1
(b) 1 and Ø
(c) C and 1
(d) CPU/RAM

15. In microcomputers, the CPU is the same as

(a) Random Access Memory.
(b) Read Only Memory.
(c) Microprocessing Unit.
(d) Keyboard.

16. The part of the microprocessing unit which contains the wiring to do mathematical functions and comparisons is called the

(a) Arithmetic-Logic Unit (ALU).
(b) Random Access Memory (RAM).
(c) Read Only Memory (ROM).
(d) The Control Unit (CU).

17. The EBCDIC has the possibility of

(a) 16K variations.
(b) 256 variations.
(c) 64K variations.
(d) 1ØØ variations.

18. The ZONE portion of each byte is

(a) the first four bits.
(b) the second four bits.
(c) the DIGIT PORTION.
(d) the first bit only.

19. In any one position of a decimal number, we can write one of ______ different values.

(a) 5
(b) 1Ø
(c) 1ØØ
(d) 16K

20. In order to get your program to work on a computer, it must first be

(a) placed into RAM.
(b) placed into ROM.
(c) placed into the CRT.
(d) placed into the printer.

WHEN YOU HAVE COMPLETED THE ENTIRE EXAM, TRANSFER YOUR ANSWERS TO THE ANSWER SHEET WHICH FOLLOWS.

ICS

ANSWER PAPER

To avoid delay, please insert all the details requested below

Subject ______________________ Course ______

Name

Address

Post Code

Serial Number	Test No.	Ed No.
2 4 7 0 2	2	2

Student's Reference

Letters / Figures

Tutor's Comments

Grade	Tutor

Study the foregoing Question Paper and use it for your rough workings. Record your final answers in the matrix below by writing a cross (X), IN INK OR BALLPOINT, through the letter which you think is the correct answer. Submit ONLY THIS ANSWER SHEET to the School for correction. ALL QUESTIONS MUST BE ANSWERED.

1.	A	B	C	D
2.	A	B	C	D
3.	A	B	C	D
4.	A	B	C	D
5.	A	B	C	D
6.	A	B	C	D
7.	A	B	C	D
8.	A	B	C	D
9.	A	B	C	D
10.	A	B	C	D
11.	A	B	C	D
12.	A	B	C	D
13.	A	B	C	D
14.	A	B	C	D
15.	A	B	C	D
16.	A	B	C	D
17.	A	B	C	D
18.	A	B	C	D
19.	A	B	C	D
20.	A	B	C	D